TRABAJO FINAL DE CARRERA

TÍTULO DEL TFC: Implementación de la capa Modelo de Elevación Digital en la plataforma ISIS

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Sistemas de Telecomunicación

AUTORA: Susana Romero Gómez

DIRECTOR: Joshua Martínez Tristancho

FECHA: 25 de Mayo 2009

Título: Implementación de la capa Modelo de Elevación Digital en la plataforma ISIS

Autora: Susana Romero Gómez

Director: Joshua Martínez Tristancho

Fecha: 25 de Mayo 2009

Resumen

Dentro de la plataforma de simulación integrada desarrollada por el grupo de investigación ICARUS de la UPC, el modelo de elevación digital juega una parte importante con la finalidad de conocer la altura respecto del suelo de los aviones no tripulados.

Este trabajo final de carrera consistirá en la implementación de la capa Modelo de Elevación Digital (DEM) en la antigua plataforma Escenario Integrado de Simulación ICARUS (ISIS).

Esta capa se abastece de dos bases de datos. Una fotográfica y otra de elevación. Dependiendo del "uso" debe ser procesada dicha información según las necesidades de la petición.

Una primera versión será integrada dentro de un visualizador 3D, llamado 'Visor', a fin de validarla. Este Visor forma parte de la antigua plataforma ISIS de simulación bajo el Framework .NET.

Palabras clave: DEM, DTM, USGD, AutoNAV4D93, ISIS, Co-Simulador, Visor 3D, Modelo digital de elevación

Title: Digital Elevation Model on ISIS platform implementation
Author: Susana Romero Gómez
Director: Joshua Martínez Tristancho
Date: May, 25th 2009

Overview

Within the integrated simulation platform developed by the ICARUS research group of the UPC, the digital elevation model plays an important part with the purpose of knowing the height from the floor of the Unmanned Aerial Vehicles.

This TFC consists on the implementation of the layer Digital Elevation Model (DEM) in the old ICARUS Simulation Integrated Scenario (ISIS).

This layer has two sources; a photographic database and a elevation database. Depending on the "use" this information must be processed according to the needs of the request.

A first version will be integrated into a 3D display called 'Visor', to validate it. This ‘Visor’ is part of the old ISIS simulation platform under the .NET Framework.

Keywords: DEM, DTM, USGD, AutoNAV4D93, ISIS, Co-Simulador, Visor 3D, Digital Elevation Model

Quiero dedicar este proyecto a mis padres y a mi hermano porque lo son todo para mí.

A Víctor, por ser el pilar fundamental de mi vida y un apoyo incondicional en todo momento.

A Silvia, que me ha acompañado en todos estos años de carrera y para mí siempre será de mi familia

Y por supuesto a mi tutor, Joshua, por todo la confianza que ha depositado en mí desde el primer día.

ÍNDICE

CAPÍTULO 1 . DEFINICIONES, GLOSARIOS Y REQUERIMIENTOS 3

1.1. El co-simulador 3

1.2. Bases científicas 3
1.2.1 Cuaterniones……..3
1.2.2 Órbitas……..……..4
1.2.3 Elementos Keplerianos……..……...5
1.2.4 Sistemas de referencia……..……...6
1.2.5 Datums……..……..……...7

CAPÍTULO 2 . TECNOLOGÍAS PARA DEM 9

2.1. Vehículos Aéreos No Tripulados (VANT) 9

2.2. USAL 9

2.3. MAREA 9

2.4. Plataforma ISIS 10

2.5. Tecnologías .NET 11

2.6. OpenGL 11

2.7. AutoNAV4D94 12

CAPÍTULO 3 . CAPA MODELO ELEVACIÓN DIGITAL 13

3.1. Definición 13

3.2. Tipos de modelos 13
3.2.1. Conjuntos de Datos Vectoriales 13
3.2.2. Conjuntos de Datos Raster 15
3.3.3. Comparativa entre modelos y toma de decisiones 15

3.3. Formatos 16
3.3.1. Modelo Digital de Elevación DTM 16
3.3.2. Modelo Digital de Elevación USGS 17

CAPÍTULO 4 . INTEGRACIÓN CON AUTONAV4D 19

4.1. Arquitectura del terreno dentro del visualizador 3D de la antigua plataforma ISIS.... 20

4.2. Ciclo de vida del terreno del visualizador 3D de la plataforma ISIS 21

4.3. Campos y métodos de las clases 22

CAPÍTULO 5 . CONCLUSIONES 25

ANEXOS……27

ÍNDICE DE FIGURAS

1.1 Representación en el eje de los cuaterniones……4
1.2 Comparativa caminos orto-loxodrómico……...5
1.3 Elementos Keplerianos……..6
1.4 Sistema ECEF……7
2.1 Primitivas del Middleware MAREA……10
3.1 Ejemplo de modelo vectorial de contornos……14
3.2 Ejemplo de modelo vectorial de red de triángulos irregulares…….14
3.3 Ejemplo de pérdida de calidad de un mapa raster……15
3.4 Ejemplo de fichero DEM……...17
4.1 Diagrama de clases……19
4.2 Arquitectura del DEM……20
4.3 Interfaz del visor……21
4.4 Funciones de entrada y salida de datos de la clase Terrain……..22
4.5 Funciones de entrada y salida de datos de la clase DEMmap……..22
4.6 Interfaz para cabeceras de DEM……22
4.7 Cabecera del formato USGS……...23
4.8 Cabecera del formato DTM……..23
5.1 Aspecto del GeoServer de la nueva plataforma ISIS……25

INTRODUCCIÓN

En el capítulo 1 se presentan algunos conceptos necesarios para entender el co-simulador donde se integrará la capa modelo de elevación digital DEM a fin de ser validada.

En el capítulo 2 se ha querido hacer una breve introducción a las tecnologías donde se integra el co-simulador *AutoNAV4D93* y algunas particularidades como la plataforma de programación o el motor de renderizado.

El capítulo 3 está dedicado al Modelo de Elevación Digital que es el contenido importante en este trabajo final de carrera. Se explicará en qué consiste así como algunos ejemplos como pueden ser el *DTM* y el *USGS* de interés para el grupo *ICARUS* en la plataforma de simulación *ISIS* explicada previamente en el capítulo 2 y que ha ido evolucionando a durante la realización de este trabajo.

El capítulo 4 hace referencia a detalles más técnicos de la implementación. Se presentará la arquitectura, el ciclo de vida del DEM y algunos diagramas de clase. Se ha dado la implementación del código en el anexo debido a su extensión de unas 1000 líneas de código nuevo añadidas a la plataforma *ISIS* dentro del co-simulador *AutoNAV4D93*. La ayuda de las nuevas clases creadas se encuentra en el propio código en formato XML de manera que pueden ser consultadas dentro del editor **Visual Studio** si usamos *Windows* o dentro del editor **Mono** si usamos *Linux*.

Este trabajo final de carrera termina con el capítulo 5 dedicado a las conclusiones, las líneas futuras y un estudio sobre el impacto ambiental.

CAPÍTULO 1 . DEFINICIONES, GLOSARIOS Y REQUERIMIENTOS

1.1. El co-simulador

En este proyecto el término co-simulador se corresponde a un grupo de componentes y servicios integrados dentro de una plataforma de ensayo en un mismo escenario de sistemas aéreos no tripulados (*Unmanned Aircraft Systems*, UAS) y sistemas simulados, que es donde vamos a probar y validar la capa de modelo de elevación digital objeto de este trabajo.

El co-simulador nos ayuda a validar el software, hardware y los planes de vuelo antes de su implementación definitiva.
Está formado por un grupo de componentes que trabajan conjuntamente y que están diseñados dependiendo de la finalidad y fiabilidad deseadas del modelo de simulación.
En nuestro caso el co-simulador se utiliza para la comprobación de diferentes misiones de vehículos aéreos no tripulados dentro de una plataforma de prueba antes de su implementación real.

En el CAPÍTULO 2 se explica detalladamente los requisitos y componentes que forman parte del co-simulador utilizado.

1.2. Bases científicas

1.2.1 Cuaterniones

Los cuaterniones nos proporcionan la notación matemática para representar las orientaciones y las rotaciones de objetos en tres dimensiones.

Permiten evitar los problemas de bloqueo en el azimut, también conocido en inglés como *gimbal-lock*, que aparecen al aplicar los ángulos de Euler donde la rotación se ve limitada por dicho efecto.
El éxito es debido a la extensión del concepto de rotación en tres dimensiones a la rotación en cuatro dimensiones.

En la siguiente figura se muestra la representación de los cuaterniones.

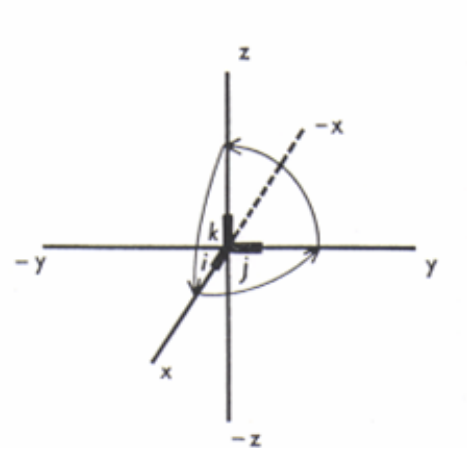


Fig. 1.1 Representación en el eje de los cuaterniones

Se representa un vector tridimensional en un sistema de coordenadas con tres ejes perpendiculares entre sí *x, y, z*, en función de los vectores unitarios *i, j, k*. La multiplicación por *i* se define como una rotación de 90 grados en el plano perpendicular al vector *i*, es decir, en el plano de *y* y *z*.

Matemáticamente definimos los cuaterniones como el conjunto:

$$\mathbb{H} = \{\mathbb{Q} = w + xi + yz + zk \; \forall \, w, x, y, z \in \mathbb{R}\} \tag{2.1}$$

Donde la suma se define como:

$$(w + xi + yj) + (a + bi + cj + dk) = (w + a) + (x + b)i + (y + c)j + (z + d)k \tag{2.2}$$

y la multiplicación se desarrolla como:

$$(w + xi + yj + zk)(a + bi + cj + dk) = \\ (wa - xb - yc - zd) + (wb + ax + yd - zc)i + (wc + ya + zb - xd)j + (xd + za + xc - yb)k \tag{2.3}$$

Definiendo las relaciones:

$$i^2 = j^2 = k^2 = ijk = -1 \; \forall \, i, j, k \in \mathbb{I} \tag{2.4}$$

1.2.2 Órbitas

En nuestro caso, la órbita nos muestra la trayectoria o el camino recto recorrido sobre la tierra.

Cuando nuestra órbita se rige por una constante, se describe el camino *loxodrómico* donde la trayectoria corta los meridianos con el mismo ángulo.

A diferencia del camino loxodrómico, el camino ortodrómico se corresponde con el camino más corto entre dos punto.

La siguientes ilustraciones muestran la comparación entre ambos.

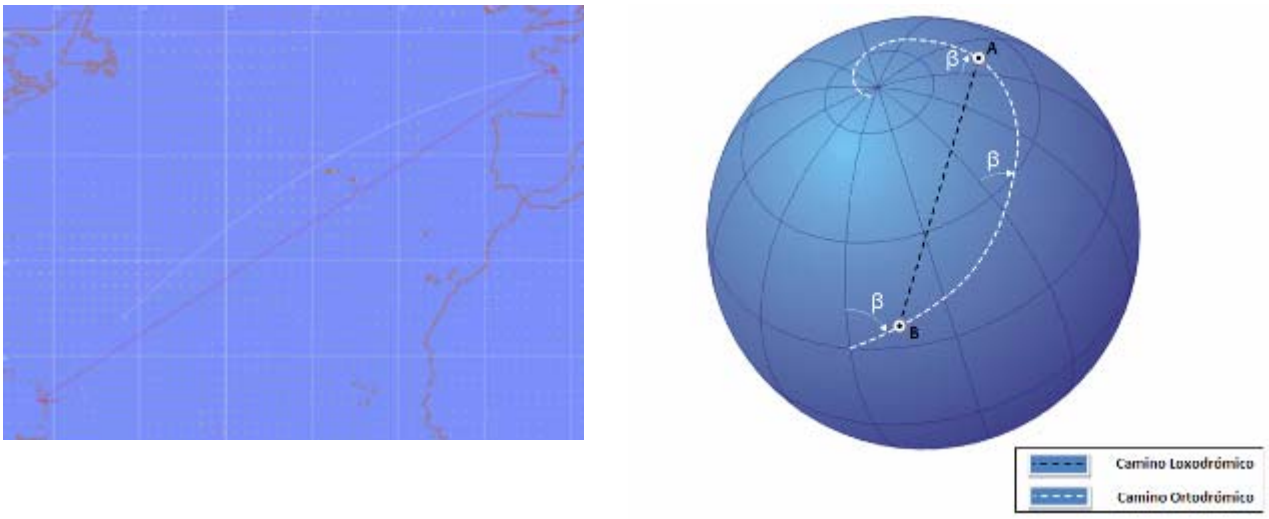


Fig. 1.2 Comparativa caminos orto-loxodrómico

1.2.3 Elementos Keplerianos

Los elementos Keplerianos nos permiten definir las órbitas de los satélites a partir de seis parámetros válidos para un instante de tiempo determinado llamado *época.*

Los tres primeros elementos proporcionan la forma de la órbita:

- **a**: semi-eje mayor de la elipse [m]
- **e**: excentricidad de la elipse
- **T**: tiempo de perigeo

Los tres restantes definen la orientación de la órbita:

- **I**: Inclinación de la órbita [rad]
- **Ω**: Longitud del nodo ascendente [rad]
- ***w**: Argumento del perigeo [rad]*

En la Fig. 1.3 se detalla cada uno de los elementos dentro de un sistema de referencia.

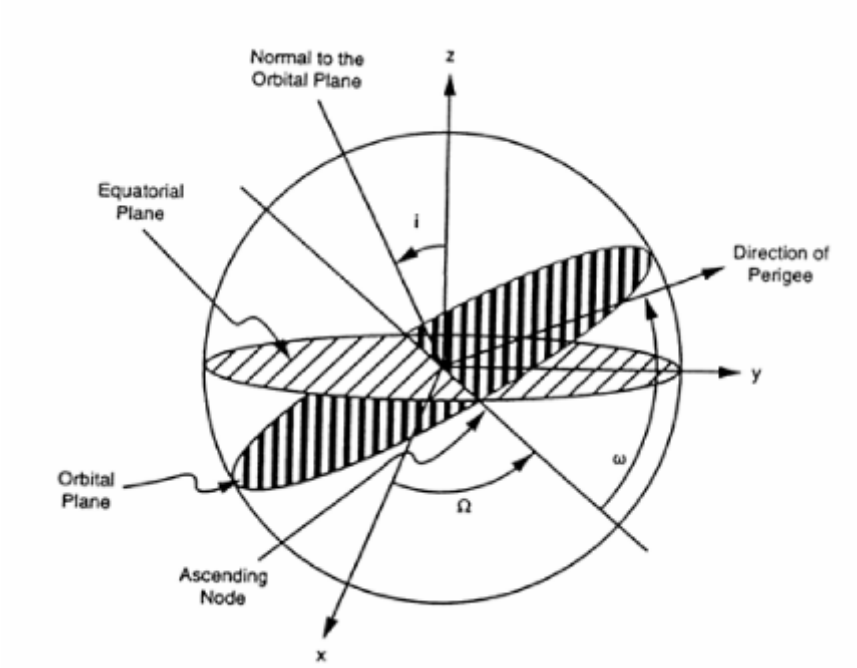


Fig. 1.3 Elementos Keplerianos

1.2.4 Sistemas de referencia

Un parámetro fundamental a definir es la posición del VANT. Para ello utilizaremos un conjunto de variables que permiten situarlo dentro de un sistema de referencia.
En el co-simulador AutoNAV4D93 se utilizan tres sistemas de referencia diferentes según las necesidades, el sistema ECEF, NED y VCVF.

El sistema denominado ECEF (Earth Centered Earth Fixed), es un sistema de coordenadas Cartesiano que, como su nombre indica, está centrado en la Tierra y rota fijado a su superficie por lo tanto sus ejes (*X,Y,Z*) son el sistema de referencia planetario evitando a su vez efectos externos como la rotación o la órbita.
El eje *Z* coincide con el eje de rotación convencional de la Tierra. El eje *X* viene dado por la intersección entre el Meridiano de Greenwich y el plano que pasa por el centro de las masas de la Tierra y es ortogonal al eje *Z*.
El eje *Y* está definido por la dirección que es perpendicular a los dos ejes anteriores tal que $Y=Z\cdot X$.

Fig. 1.4 Sistema ECEF

Este primer sistema de referencia se utiliza para geo-posicionar un vehículo en coordenadas Cartesianas.

Para poder definir interacciones locales con el centro de gravedad del vehículo utilizaremos el sistema de referencia ortogonal NED (North East Down).
El eje *Z* siempre señala el centro de la tierra, el eje *X* el Norte terrestre y el eje *Y* el Este.
Existe una versión de este sistema de referencia llamada ángulos de Euler que usa coordenadas esféricas. Usaremos este sistema de referencia compatible con los simuladores de vuelo.

El sistema VCVF (Vehicle Centered Vehicle Fixed) es el sistema de referencia del vehículo.
El eje *i* siempre va en dirección de la cabeza del vehículo que normalmente se corresponde con la dirección de vuelo. El eje *j* apunta a la parte derecha del vehículo y *k* siempre apunta a la parte superior.
Al utilizar estos ejes ortogonales omitimos los movimientos internos como desplazamientos de partes del vehículo y rotaciones.

1.2.5 Datums

Todos sabemos que la tierra no es esférica. Pero, no solo eso, ni siquiera es un cuerpo regular achatado por los polos. Esta irregularidad hace que cada país, o incluso cada región, escoja el modelo de cuerpo (definible matemáticamente) que más se ajuste a la forma de la tierra en su territorio. Este cuerpo suele ser un elipsoide.
Una vez que hayamos sido capaz de definir el elipsoide, necesitaremos crear otro modelo matemático que nos permita representar un punto concreto en un mapa con sus valores de coordenadas. A este modelo matemático le llamamos *Datum*. Ahora hay que destacar que un *Datum* tiene que estar necesariamente referido a un elipsoide en particular: Éste es conocido como el **elipsoide de**

referencia*.* La conclusión obvia es que existen diferentes *Datums* si se usan diferentes elipsoides de referencia.

El *Datum* WGS84 es el *Datum* de referencia universal utilizado en aplicaciones GPS y en Vehículos Virtuales. No obstante el visualizador utiliza el *Datum* local de la posición donde se encuentre por ejemplo, si un vehículo se encuentra en la ciudad de Barcelona, se usará el *Datum* ED50 utilizado en España y Portugal.

CAPÍTULO 2 . TECNOLOGÍAS PARA DEM

2.1. Vehículos Aéreos No Tripulados (VANT)

Un Vehículo Aéreo No Tripulado (en inglés UAV U*nmanned Aerial Vehicle*) es un vehículo aéreo autónomo, capaz de volar sin necesidad de piloto humano, gracias a un sistema de pilotaje autónomo.

Son de gran utilidad en contextos donde sea peligroso el uso de aviones tradicionales, en ambientes hostiles o misiones repetitivas como la vigilancia y el reconocimiento.

2.2. USAL

El *UAS Service Abstraction Layer* (USAL), desarrollado por el equipo ICARUS [1], es un conjunto de servicios disponibles dentro de la arquitectura del sistema del VANT para dar soporte a las misiones del vehículo y facilitar el acceso de los parámetros al usuario final. Los servicios disponibles de USAL se clasifican en cuatro categorías:

- Servicios de vuelo
- Servicios de misión
- Servicios de carga
- Servicios de reconocimiento

2.3. MAREA

El flujo de datos entre los componentes de USAL y los servicios se realiza a través del *middleware* MAREA (*Middleware Arquitecture for Remote Embedded Applications*) desarrollado por el equipo ICARUS para la comunicación de sus primitivas formadas por variables, eventos, invocaciones remotas y transferencia de archivos.
MAREA provee un entorno de ejecución con canales de comunicación y funciones comunes.

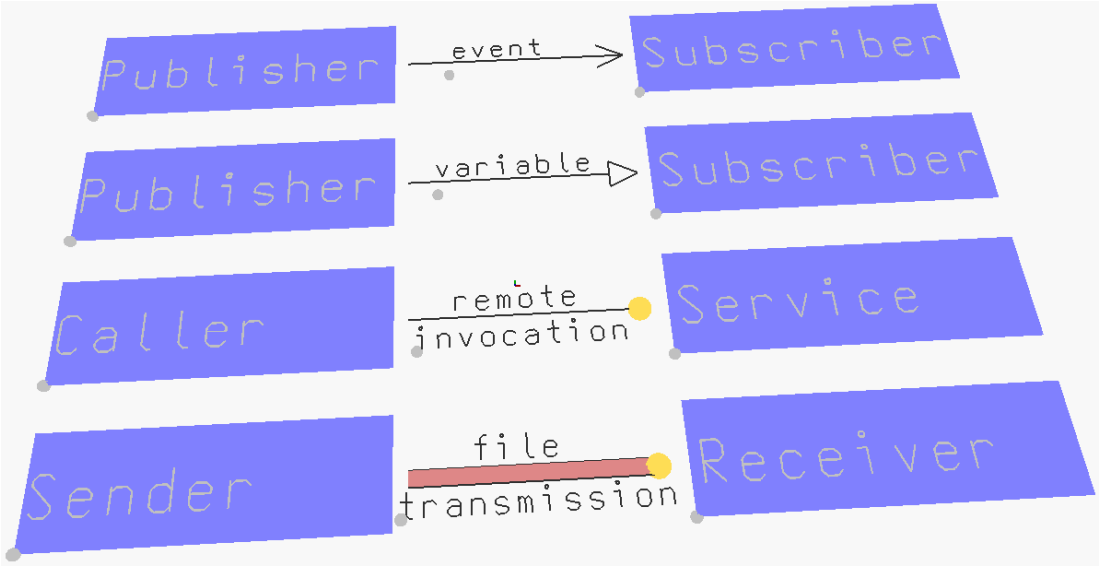


Fig. 2.1 Primitivas del *Middleware* MAREA

El papel de cada servicio se representa por la acción de publicar, subscribir o ambas acciones simultáneamente, lo que facilita la implementación en servicios empotrados eliminando la programación compleja de redes de aplicaciones distribuidas.
Este *Middleware* ofrece la localización de otros servicios y administra su descubrimiento en la red, maneja todas las tareas de transferencia, direcciona los mensajes y los retransmite, entrega los datos, controla el flujo de los mismos, etc. También es un contenedor de servicios cooperativos.

Durante la realización de este TFC el *Midlleware* MAREA ha ido evolucionando. Al inicio sólo disponía de la primitiva *variable* y el *Visor* está basado en la versión 1.1. Actualmente MAREA dispone de las cuatro primitivas. En un futuro será necesario replantear la arquitectura del Visor. De mientras se usa un servicio llamado *GoogleEarthDisplayService* que hace la función de lo que será el *GeoServer* o proveedor de mapas

2.4. Plataforma ISIS

La plataforma ISIS (*Icarus simulation Integrated Scenario*) es un conjunto de servicios reusables que comprenden una serie mínima de componentes comunes necesarios en la mayoría de las misiones de vehículos aéreos no tripulados.
La idea es proveer una capa de abstracción que permita al desarrollo de la misión reutilizar estos componentes y que proporcione una serie de directrices de cómo los servicios deben intercambiar la información de los aviones de cualquier otro tipo de información.
Los servicios disponibles cubren una parte importante de las funcionalidades genéricas que se presentan en la mayoría de las misiones. Por lo tanto, para

adaptar el avión a una nueva misión será necesario reconfigurar los servicios utilizados en el mismo.

El objetivo de ISIS es minimizar la prueba del desarrollo y el coste de validación, y proporcionar una fácil migración del *software* desde el entorno de pruebas hasta la plataforma real de vuelo.

2.5. Tecnologías .NET

El co-simulador debe de ser un sistema distribuido por lo tanto debe crearse como una serie de componentes separados para ser reutilizados en otras implementaciones.

El lenguaje de programación C# es el lenguaje de programación orientado a objetos más deseado. Los programas *Visual Studio* 2005 para *Windows* y *Mono* para *Linux* soportan este lenguaje.
Los servicios se ejecutan en un 'Container'. Existen containers que sólo son soportados por C, como PSoC (*Programable System on Chip)* o las FPGAS (*Fiel Programable Gate Array*). Otros containers usados comúnmente en aplicaciones de vehículos aéreos no tripulados soportan *.NET Framework 2.0* los cuales son los containers de interés para esta aplicación.

.NET Framework 2.0 es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permite un rápido desarrollo de aplicaciones. Debido a la publicación de la norma para la infraestructura común de lenguajes (*CLI* por sus siglas en inglés), el desarrollo de lenguajes se facilita, por lo que el marco de trabajo .NET soporta ya más de 20 lenguajes de programación y es posible desarrollar cualquiera de los tipos de aplicaciones soportados en la plataforma con cualquiera de ellos, lo que elimina las diferencias que existían entre lo que era posible hacer con uno u otro lenguaje.

2.6. OpenGL

El co-simulador puede mostrar información de manera visual como contadores, el plan de vuelo, cámaras virtuales de vuelo, entre otras. Toda esta información se mostrará gracias al uso de *OpenGL* (*Open Grafic libraries*) soportado tanto por *Visual Studio* como por *Mono*.

OpenGL es una especificación estándar que define una *API* (*Application Programming Interface*) multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.
Multiplataforma es una interfaz entre dos implementaciones; en nuestro caso es una interfaz entre la implementación del software del co-simulador y la implementación hardware de la tarjeta de video.

2.7. AutoNAV4D94

Como explica el capítulo 1.1, un co-simulador es el que nos permite simular en un mismo escenario sistemas aéreos no tripulados reales y sistemas simulados.

El co-simulador empleado en este proyecto es AutoNAV4D93. El uso de este co-simulador nos da la capacidad de prototipado rápido y simulación a la capa de de servicios USAL a través de estándares y componentes que permiten un rápido diseño e implementación de nuevas funcionalidades.
La simulación se usa para incrementar la seguridad al usar vehículos virtuales y reducir el coste de diseño.

Su arquitectura está formada por tres componentes:

- Vehículos Virtuales (VV, Virtual Vehicle)
- Sistema Virtual (VS, Virtual System) y
- la interfaz del usuario

Para más detalle se recomienda ver "*AutoNAV4D. A co-simulator for Unmanned Aircraft Systems*" [2].

CAPÍTULO 3 . CAPA MODELO ELEVACIÓN DIGITAL

3.1. Definición

El Modelo de Elevación Digital (DEM) es una estructura numérica de datos que representa la superficie topográfica terrestre y la distribución espacial de la altitud.
Un terreno real puede describirse de forma genérica como una función bivariable continua:

$$z = \zeta (x,y) \tag{3.1}$$

donde *z* representa la altitud del terreno en el punto de coordenadas *(x, y)* y ζ es una función que relaciona la variable con su localización geográfica.
En un modelo digital de elevaciones se aplica la función anterior sobre un dominio espacial concreto, *D*. En consecuencia, un DEM puede describirse genéricamente como:

$$DEM = (D, \zeta) \tag{3.2}$$

En la práctica, la función no es continua sino que se resuelve a intervalos discretos, por lo que el DEM está compuesto por un conjunto finito y explícito de elementos. Los valores de x e y suelen corresponder con las abscisas y ordenadas de un sistema de coordenadas plano, habitualmente un sistema de proyección cartográfico.

3.2. Tipos de modelos

Como se deduce del apartado anterior, la unidad básica de información en un DEM es un punto acotado compuesto por un valor de altitud *z* con el valor correspondiente de *x* e *y.* Las estructuras de datos correspondientes a los modelos digitales de terreno se definen dentro de tres conjuntos diferenciados, conjuntos de Datos Vectoriales, Datos Raster y Datos alfanuméricos.
Nos centraremos en los dos primeros conjuntos, vectoriales y *raster* ya que el último (datos alfanuméricos) se corresponde con los atributos de interés, coordenadas, nombres topográficos etc. que complementan a los demás modelos.

3.2.1. Conjuntos de Datos Vectoriales

Se basa en entidades u objetos geométricos definidos por las coordenadas de sus nodos y vértices.
Se corresponden a la presentación digital de los mapas mediante una representación de puntos, líneas y líneas que conforman áreas. Esos datos se

encuentran separados por temas en diferentes capas de información tales como vías de comunicación, localidades, hidrografía, curvas de nivel, etc.
Los puntos se definen mediante un par de valores de coordenadas con un atributo de altitud, las líneas mediante un vector de puntos - de altitud única o no- y los polígonos mediante una agrupación de líneas.
Los modelos vectoriales se pueden expresar según estructuras de contorno o redes de triángulos irregulares (TIN).
Las primeras forman el DEM a partir de un conjunto de curvas de nivel que pasan por la zona representada, separadas generalmente por intervalos constantes de altitud, más un conjunto de puntos acotados que definen lugares singulares (cimas, fondos de colinas, collados, etc.).

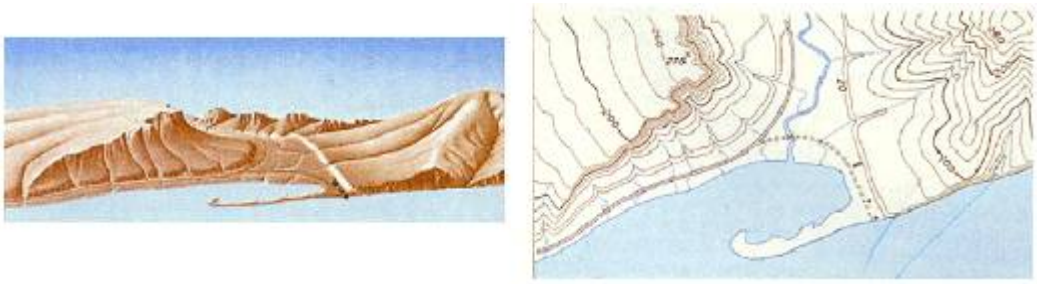


Fig. 3.1 Ejemplo de modelo vectorial de contornos

En las redes de triángulos irregulares (*TIN, triangulated irregular network*) los triángulos se construyen ajustando un plano a tres puntos cercanos no co-lineales, y se adosan sobre el terreno formando un mosaico que puede adaptarse a la superficie con diferente grado de detalle, en función de la complejidad del relieve.

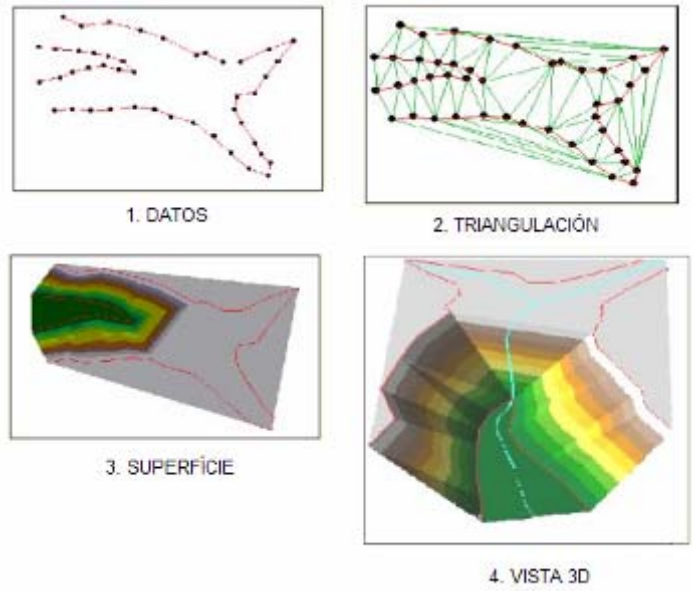


Fig. 3.2 Ejemplo de modelo vectorial de red de triángulos irregulares

3.2.2. Conjuntos de Datos Raster

En el modelo *raster*, los datos se interpretan como el valor medio de unidades elementales de superficie no nula que teselan el terreno con una distribución regular, sin solapamiento y con recubrimiento total representada. Estas unidades fundamentales son los *píxeles* que guardan información de la altura, anchura y color del mismo.
Los gráficos rasterizados se distinguen de los gráficos vectoriales en que estos últimos representan una imagen a través del uso de objetos geométricos como curvas y polígonos, no del simple almacenamiento del color de cada píxel.

Fig. 3.3 Ejemplo de pérdida de calidad de un mapa *raster*

3.3.3. Comparativa entre modelos y toma de decisiones

A continuación se muestran algunas ventajas y desventajas que tiene un modelo sobre el otro:

1) El modelo *Raster* es más eficiente en cuanto a cálculos que el modelo vectorial. La distribución regular permite que las diferentes transformaciones (rotación, escalado, etc.) puedan realizarse eficientemente mediante la implementación de algún algoritmo incremental, mientras que una malla de triángulos, para el caso del modelo vectorial, al no tener una construcción uniforme hace más complicados dichos cálculos.

2) El modelo *Raster* es más eficiente en memoria que le modelo vectorial. Habitualmente siempre se necesitan más datos para su reproducción.

3) El modelo Vectorial es más exacto que el modelo *Raster*, permitiendo un mayor grado de detalle. Los mapas vectoriales poseen la cualidad

de poder ser escalados ilimitadamente sin sufrir pérdida de calidad a diferencia de los mapas pasterizados.

En el caso particular del proyecto que nos ocupa, el modelo *Raster* tiene mayor validez por las características citadas anteriormente.
Tiene una mayor eficiencia computacional sacrificando a la vez la calidad del detalle visual que no entra dentro de los objetivos de la aplicación a desarrollar.

3.3. Formatos

Este apartado trata de abordar los principales formatos de datos para las aplicaciones donde esté presente el Modelo Digital de Elevaciones.

3.3.1. Modelo Digital de Elevación DTM

Los modelos digitales de terreno DTM [3] (Digital Terrain Model) se generan a partir del procesado ortofotos.
Una ortofoto es una fotografía de una zona de la superficie terrestre que ha sido corregida digitalmente y representa una proyección ortogonal de la zona sin efectos de perspectiva permitiendo realizar mediciones exactas sobre el terreno sin que influyan las deformaciones causadas por la perspectiva de una cámara o la velocidad a la que se mueve.

El formato de los ficheros DTM es el formato '*grid ASCII*'. Los datos que contiene están formados por una cabecera seguida del conjunto de los datos.

La figura 3.4 muestra un ejemplo de fichero DTM que nos permite ver la estructura de sus datos:

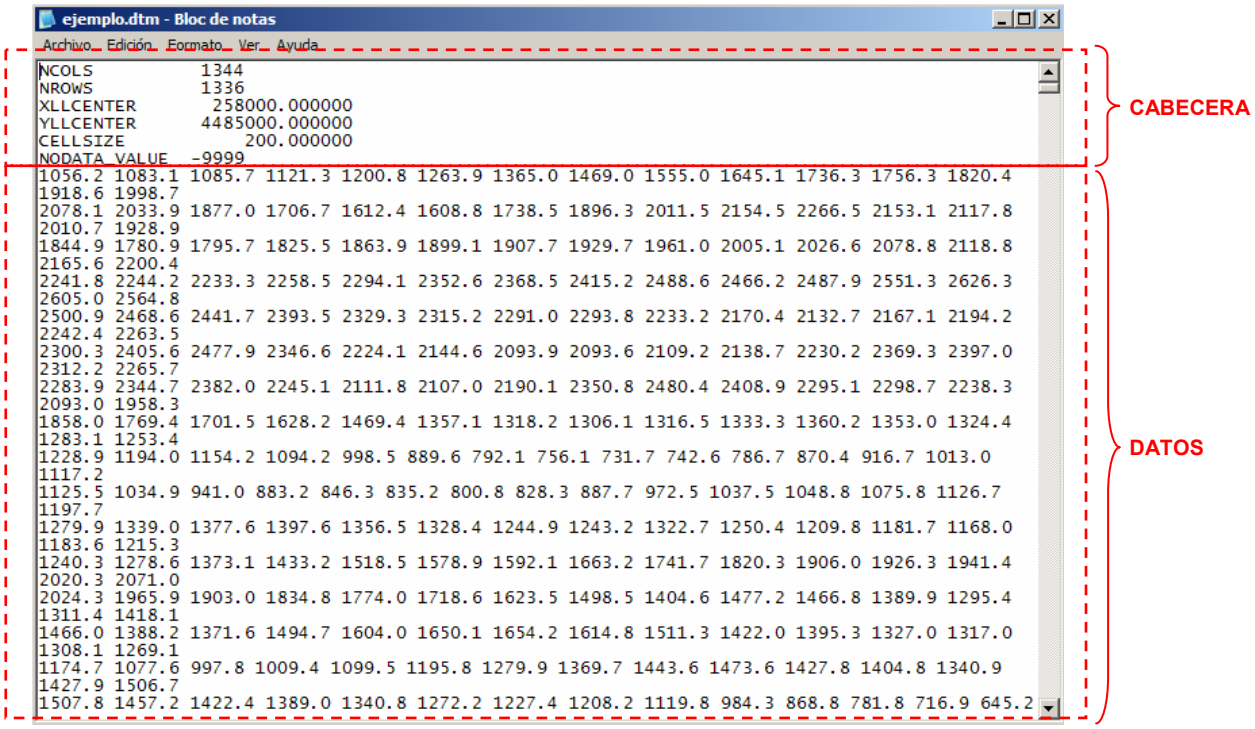


Fig. 3.4 Ejemplo de fichero DEM

La cabecera nos da información sobre el contenido y características de los datos del DTM. Estos datos nos serán de gran utilidad para poder almacenar el fichero y tratarlo correctamente.
Indica el valor del número de filas y columnas de los datos, las coordenadas del centro del píxel sur-oeste, el tamaño del píxel para cada dato y el valor con el que se expresa una cota inexistente.

A continuación encontramos las cotas ordenadas por filas de Norte a Sur y cada fila de Oeste a Este. Las cotas vienen dadas en metros y separadas por blancos (espacios).

Los datos contenidos en este fichero se encuentran referenciados en el sistema oficial ED50 proyección UTM huso 31 y las cotas son ortonométricas.
La altura ortonométrica es la distancia en vertical entre la superficie física del planeta y la superficie del geoide.

3.3.2. Modelo Digital de Elevación USGS

El estándar USGS DEM es un formato de archivo desarrollado por la United States Geological Survey [4] para el almacenamiento de datos *Raster* del modelo de elevación digital. Es un estándar abierto utilizado mundialmente.
Está contenido en un único archivo codificado en ASCII formado por bloques de 1024 bytes registrado en tres categorías, A, B y C.

El registro A se corresponde a la cabecera del archivo, el C aparece en la cola como tráiler y múltiples registros de B, llamados retratos, comprenden los datos de elevación. Los registros de A y C están formados por un solo bloque mientras que B normalmente requiere múltiples bloques.

Los campos del registro A contienen el origen, el tipo y un resumen de los datos.

En el registro B se encuentran todos los datos de elevación para un lugar en concreto. Los registros son múltiples de 1024 bytes y contienen una pequeña cabecera con un resumen de la información de ese mismo registro. La información de esta cabecera será de gran utilidad en nuestra implementación para poder dar forma a nuestro mapa DEM correctamente.
Las elevaciones son contiguas y están separadas por '*voids*' de valor -32767. Cada elevación se expresa con seis caracteres enteros que ocupan un lugar fijo dentro de un bloque. Los valores de este formato utilizan el sistema terrestre de referencia UTM.
La cabecera solamente aparece en le primer bloque seguido de todos los demás bloques que contienen los valores de elevación.

Por último, el registro C contiene datos de control de calidad RMSE (*root-mean squared error*) utilizando campos de 6 enteros.

Cuando leemos el fichero DEM desde el primer hasta el último byte, recorremos los valores siguiendo el orden de las columnas de oeste a este.

CAPÍTULO 4 . INTEGRACIÓN CON AutoNAV4D

Para la implementación de la capa Modelo Digital de Elevación (DEM) en la antigua plataforma Escenario Integrado de Simulación ICARUS (ISIS), hemos pensado en crear una clase llamada *DEMmap* que abstrae el modelo digital de elevación del terreno de la fuente de información.

Como vemos en la figura 4.1, el *VisorControl* que es un control de usuario, contiene una serie de añadidos (*Addons*) como pueden ser capas de dibujo (*Layer*), cámaras de puntos de vista (*Camera*) y una de ellas es el terreno sobre el que se encuentra la cámara (*Terrain*).

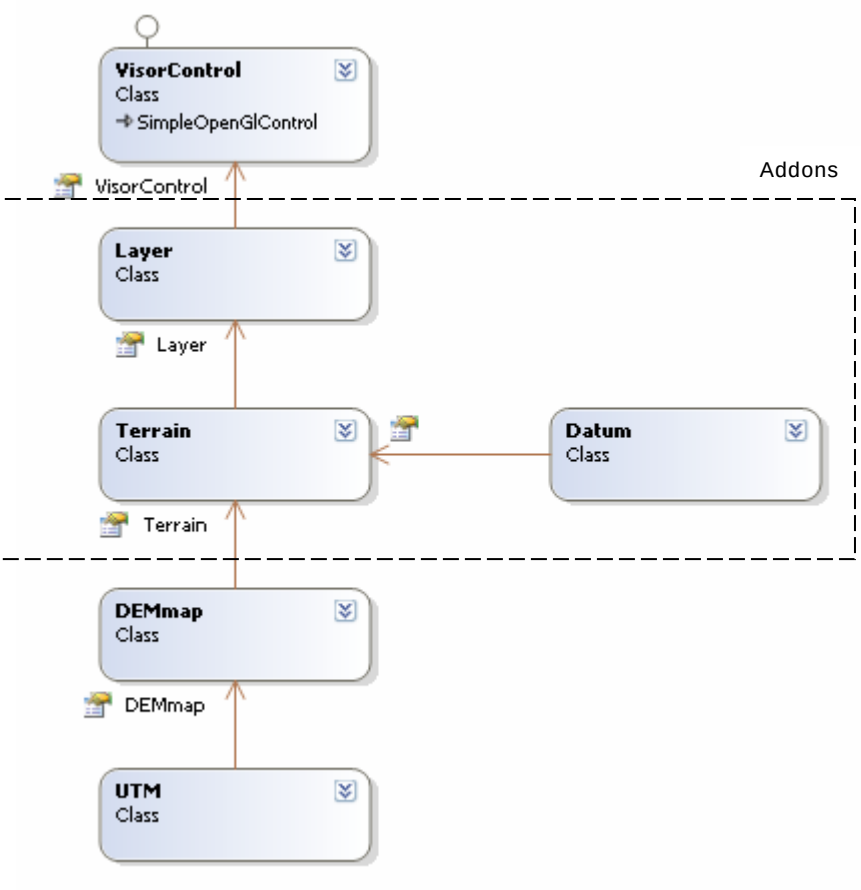


Fig. 4.1 Diagrama de clases

La clase *Terrain* que se encarga de dibujar en el terreno, implementa el modelo de elevación de terreno con el nombre de *map*. En la figura 4.2 podemos ver que las fuentes de información tienen formato USGS o DTM, para ello se ha creado una Interfaz para poder añadir fácilmente otros formatos de modelo digital de elevación. Podemos ver que todos los formatos que proveen el modelo digital de elevación consisten en una matriz de puntos cuyas coordenadas habrá que transformar a UTM según el caso.

4.1. Arquitectura del terreno dentro del visualizador 3D de la antigua plataforma ISIS

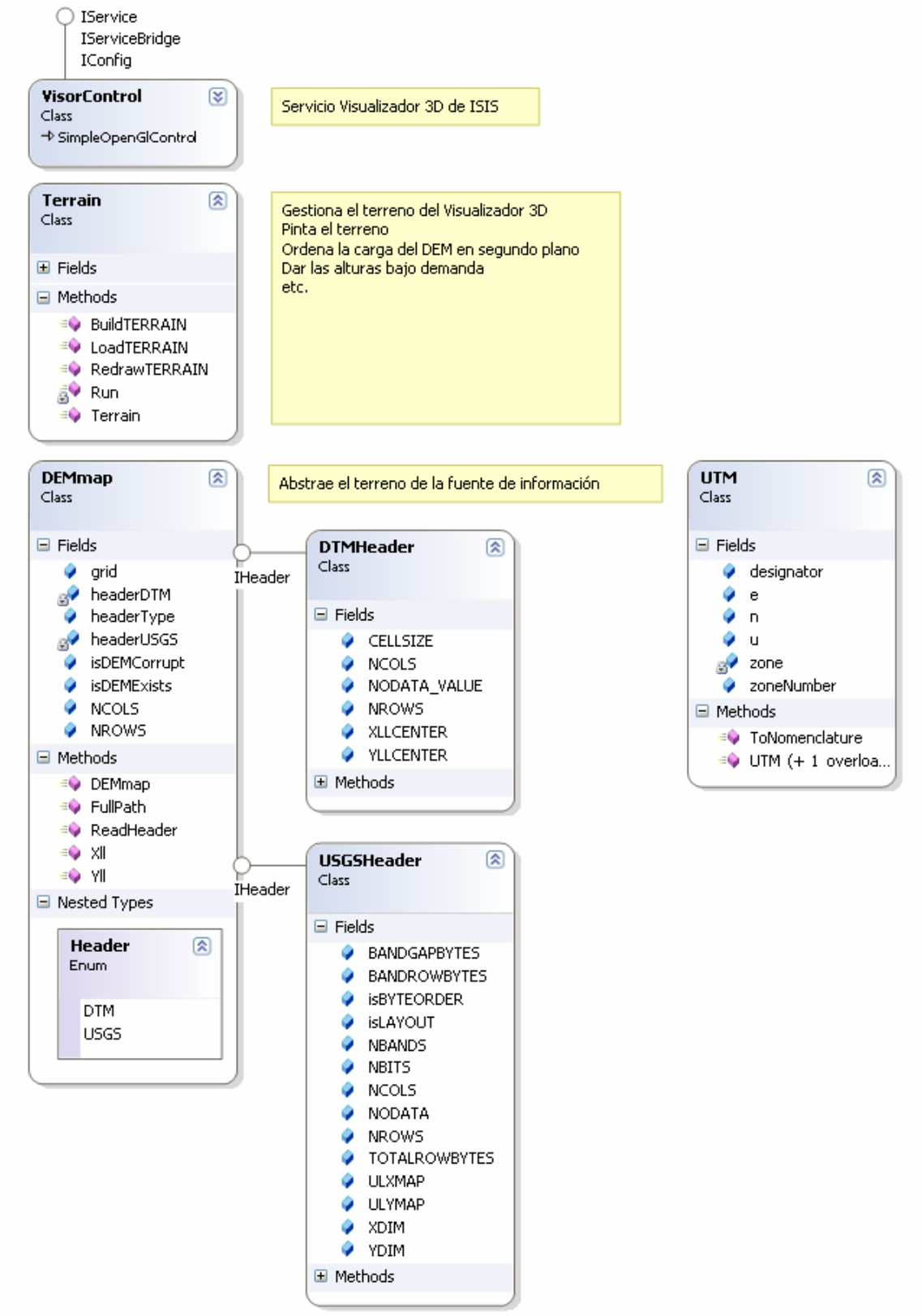


Fig. 4.2 Arquitectura del DEM

4.2. Ciclo de vida del terreno del visualizador 3D de la plataforma ISIS

El visualizador 3D de ISIS solicita la carga del DEM en función del punto de vista de la cámara. Mientras tanto el visualizador 3D muestra una pequeña zona del terreno inmediato a la cámara como se observa en la figura 4.3. El DEM se carga en segundo plano de la fuente de información según el formato deseado. Cuando el DEM finaliza el proceso de carga avisa al visualizador 3D y este junta la textura con el modelo digital de elevación.
El resultado de la unión de la textura y el DEM podrá ser utilizado en un futuro por cualquier servicio, incluido el visualizador 3D como es en nuestro caso. Existen librerías que hacen este proceso como es el caso de SharpMaps, ESRI, etc. Pero que son pesadas y consumen muchos recursos.

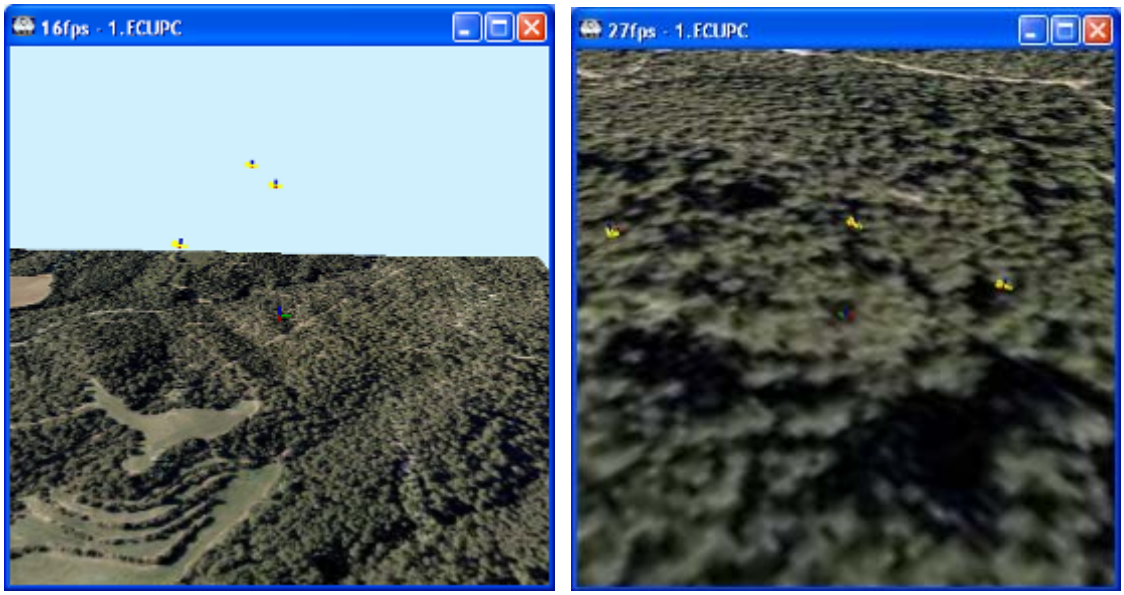


Fig. 4.3 Interfaz del visor

4.3. Campos y métodos de las clases

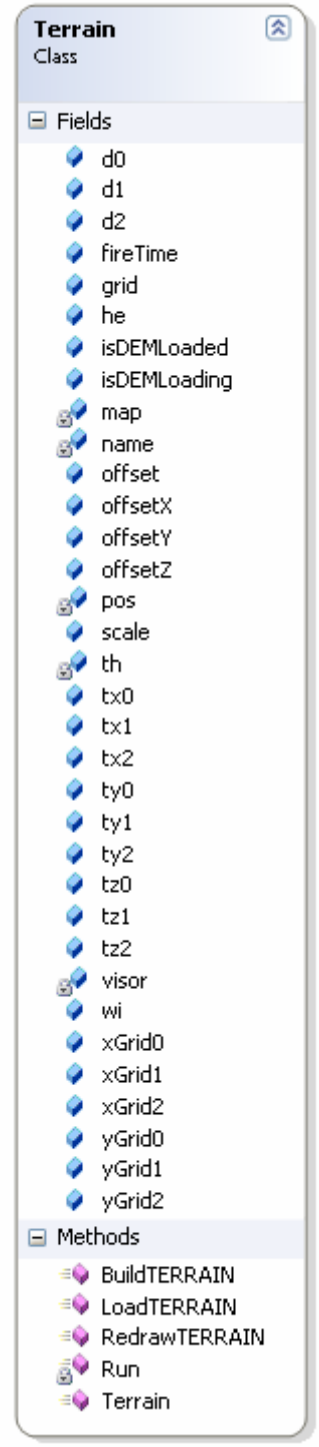


Fig. 4.4 Funciones de entrada y salida de datos de la clase *Terrain*

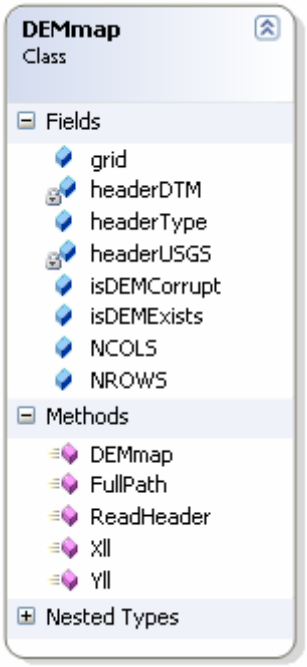


Fig. 4.5 Funciones de entrada y salida de datos de la clase *DEMmap*.

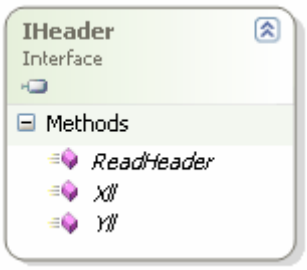


Fig. 4.6 Interfaz para cabeceras de DEM

4.3.1. Cabeceras de DEM

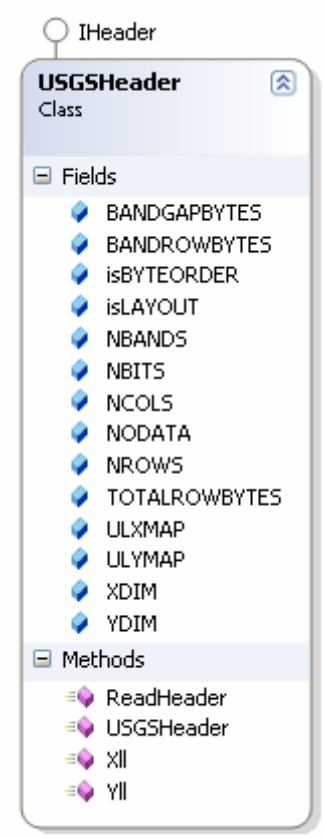


Fig. 4.7 Cabecera del formato USGS

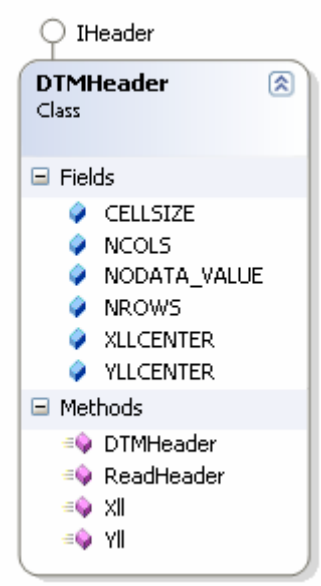


Fig. 4.8. Cabecera del formato DTM

CAPÍTULO 5 . CONCLUSIONES

Los formatos de DEM, el *DTM* y el *USGS* son de interés para el grupo *ICARUS* en la plataforma de simulación *ISIS*. Se ha provisto de unas clases selladas capaces de cargar en la memoria local y en forma de matriz, la elevación del terreno a partir de ficheros en dichos formatos. La plataforma de simulación ISIS ha estado evolucionando durante la realización de este trabajo. La validación del DEM en la antigua plataforma permitirá definir una lista de especificaciones más realistas para el modelo de elevación digital definitivo.

LINEAS FUTURAS

- Para la escala global se usa se usará la USGS que no está completado.
- Este proceso lo utiliza el visualizador 3D pero en un futuro lo podrá usar cualquier servicio de la nueva plataforma ISIS una vez Marea implemente las llamadas remotas.
- La carga de ficheros es local pero en un futuro hay que utilizar el *GeoServer* (figura 5.1) que es quien proporciona la información geográfica en la nueva plataforma ISIS.
- Un futuro servicio de altímetro podría utilizar esta capa de Modelo de Elevación Digital.

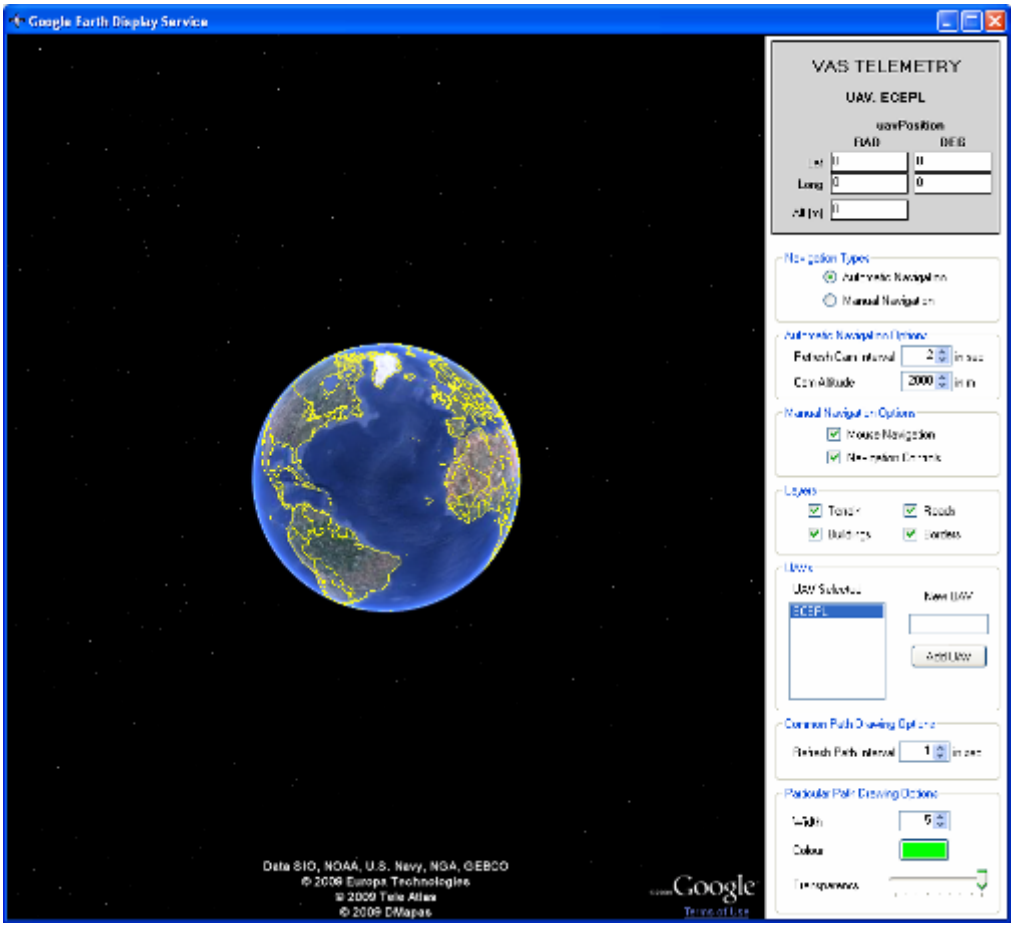


Fig. 5.1 Aspecto del GeoServer de la nueva plataforma ISIS

IMPACTO AMBIENTAL

Una vez finalizado el proyecto podemos evaluar el impacto ambiental que puede provocar la implementación de todos los sistemas que componen la plataforma de simulación.

Se entiende como impacto ambiental a las acciones humanas que pueden provocar efectos colaterales sobre el medio natural o social.

El uso de recursos informáticos contribuye al consumo de electricidad y la emisión de carbono a la atmosfera pudiéndose reducir con el empleo de técnicas de ahorro de energía, optimizando las aplicaciones a utilizar o aumentando la eficiencia mediante procesadores más veloces.

Por otro lado, el uso de la plataforma ISIS permite probar un nuevo modelo de UAV o una nueva misión, así como su impacto dentro de cada operación de vuelo sin necesidad de usar un vuelo real.
De esta manera no solo ahorramos recursos si no que la operación real es más segura ya que ha sido simulada en varias ocasiones antes de ejecutarla en la realidad.
Ahorramos combustible, horas de vuelo contaminando menos el medio ambiente y realizando con mayor eficiencia las misiones a realizar.

Debido a que una de las finalidades de la plataforma ISIS del grupo ICARUS es la monitorización de incendios y puntos calientes o HOT-SPOTS, la implementación del DEM de una forma segura y validada es crucial para que podamos por un lado combatir el fuego y eliminar que el fuego se vuelva a avivar y por el otro lado que el trabajo de los bomberos sea más seguro. Esta es otra forma de contribuir a la reducción del impacto ambiental.

REFERENCIAS Y ENLACES

[1] Grupo de investigación ICARUS de la UPC.

http://icarus.upc.es/

[2] Tristancho, Joshua. *AutoNAV4D. A co-simulator for Unmanned Aircraft Systems*.
http://upcommons.upc.edu/handle/123456789/133773

[3] *DTM. Digital Terrain Model format.*

http://www.ew.govt.nz/Environmental-information/REDI/900127/

http://www.icc.es/web/content/ca/prof/cartografia/descarrega.html

http://www.mass.gov/mgis/dtm.htm

[4] *USGS. Standards for Digital Elevation Models.*

http://rockyweb.cr.usgs.gov/nmpstds/demstds.html

[5] Información sobre mapas *raster* y vectoriales

http://www.udistrital.edu.co/comunidad/profesores/rfranco/vector_raster.htm#estructuraraster

[6] *Ortofotos*

http://www.cartesia.org/article.php?sid=409

ANEXOS

TÍTULO DEL TFC: Implementación de la capa Modelo de Elevación Digital en la plataforma ISIS

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Sistemas de Telecomunicación

AUTORA: Susana Romero Gómez

DIRECTOR: Joshua Martínez Tristancho

FECHA: 25 de Mayo 2009

VisorControl.Addons.Terrain

```
    public Terrain(VisorControl visor)
    public void BuildTERRAIN()
    public void RedrawTERRAIN()
    public void LoadTERRAIN(string fileName)
    private void Run()
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using Tao.OpenGl; // Open Graphic Library


namespace VisorControl.Addons
{
    /// <summary>[PENDING] The Local Terrain class. 'Camera' and 'CameraManager'
required</summary>
    /// <remarks>
    /// <para>Addons/Terrain. AutoNAV4D 4.1 Revisión 93</para>
    /// <para>©Susana Romero</para>
    /// <para>Esta clase está protegida por la licencia BSD de código abierto.</para>
    /// <para>0.Puedes ejecutar el programa con cualquier propósito.</para>
    /// <para>1.Puedes estudiar y/o modificar el programa.</para>
    /// <para>2.Puedes copiar el programa de manera que puedas ayudar a tu
vecino.</para>
    /// <para>3.Puedes mejorar el programa, y hacer públicas las mejoras, de forma que
se beneficie toda la comunidad.</para>
    /// <para>...</para>
    /// <para>...</para>
    /// <para>...</para>
    /// <para>MANUAL DE USUARIO</para>
    /// <para>...</para>
    /// <para>IMPORTAR UNA IMAGEN</para>
    /// <para>Para que 'Terrain' muestre una imagen debe estar en la carpeta '/Visor' en
formato PNG y con la propiedad 'Copiar en el directorio de resultados' como 'Copiar
siempre'. El canal 'alpha' sirve para darle transparencia a la textura, por eso deben
tener el canal 'alpha' a negro porque si no la imagen sale blanca.</para>
    /// <para>El procedimiento para que las imágenes creadas con 'Paint' tengan canal
'alpha' consiste en abrir la imagen con el 'Gimp2'. El 'Gimp' es un editor de imagenes
GNU. Al darle a menu 'Capa/Duplicar Capa' aparece el canal 'alpha' por defecto opaco.
Hay que 'Guardar' la imagen, donde en el menu para 'Exportar archivo' debemos elegir
'Combinar las capas visibles' y darle al botón 'Exportar'.</para>
    /// <para>...</para>
    /// <para>...</para>
    /// </remarks>
    [Serializable]
    public class Terrain
    {
        #region Fields
        VisorControl visor; // Access to the Visor
        DEMmap map; // Mapa que contiene el DEM
        /// <summary>[PENDING] Asyncronous processing</summary>
        private Thread th;
        string name; //visor.Path + Groups/DEM/ + fileName + .DTM
        /// <summary>[PENDING] Fire time</summary>
        public DateTime fireTime; // Hora del terreno
        /// <summary>[PENDING] Terrain grid</summary>
        public double[] grid; // Terreno cerca de Castellar
        int pos;
        /// <summary>[PENDING] Scale terrain</summary>
        public double scale; // Escala del terremo
        /// <summary>[PENDING] Terrain offset</summary>
        public double offsetX;
        public double offsetY;
        public double offsetZ;
        /// <summary>[PENDING] Terrain grid width less one</summary>
        public double wi; // Ancho de la matriz - 1
        /// <summary>[PENDING] Terrain grid height less one</summary>
        public double he; // Alto de la matriz - 1
        /// <summary>[PENDING] True when DEM is loaded in this Terrain</summary>
        public bool isDEMLoaded; // True cuando este Terrain tiene cargado un DEM
        /// <summary>[PENDING] True when Terrain is Loading a DEM</summary>
```

```
        public bool isDEMLoading; // True cuando este Terrain está cargando un DEM
        public double offset; // Desplazamiento respecto al lado
        /// <summary>[PENDING] The nearest terrain grid point</summary>
        public double d0; // El primero más cercano
        public double tx0;
        public double ty0;
        public double tz0;
        public double xGrid0; // Coordenada X a que corresponde en la rejilla
        public double yGrid0; // Coordenada Y a que corresponde en la rejilla
        /// <summary>[PENDING] The second nearest terrain grid point</summary>
        public double d1; // El segundo más cercano
        public double tx1;
        public double ty1;
        public double tz1;
        public double xGrid1; // Coordenada X a que corresponde en la rejilla
        public double yGrid1; // Coordenada Y a que corresponde en la rejilla
        /// <summary>[PENDING] The third nearest terrain grid point</summary>
        public double d2; // El tercero más cercano
        public double tx2;
        public double ty2;
        public double tz2;
        public double xGrid2; // Coordenada X a que corresponde en la rejilla
        public double yGrid2; // Coordenada Y a que corresponde en la rejilla
        #endregion Fields

        public Terrain(VisorControl visor)
        {
            this.visor = visor;
            map = new DEMmap();
            pos = visor.cManager.Current();
            scale = 1.0;
            offsetX = 0.0;
            offsetY = 0.0;
            offsetZ = -1.0;
            wi = 10.0 - 1.0; // Ancho de la matriz - 1
            he = 10.0 - 1.0; // Alto de la matriz - 1
            isDEMLoaded = false; // Este terrain no tiene aun un mapa cargado
            isDEMLoading = false; // Aún no se ha cargado el DEM
            offset = -(double)((int)((wi + 1.0) / 2.0)); // Desplazamiento respecto al
lado
            d0 = visor.camera.cRange; // El primero más cercano
            tx0 = 0.0;
            ty0 = 0.0;
            tz0 = 0.0;
            xGrid0 = 0.0;
            yGrid0 = 0.0;
            d1 = visor.camera.cRange; // El segundo más cercano
            tx1 = 0.0;
            ty1 = 0.0;
            tz1 = 0.0;
            xGrid1 = 0.0;
            yGrid1 = 0.0;
            d2 = visor.camera.cRange; // El tercero más cercano
            tx2 = 0.0;
            ty2 = 0.0;
            tz2 = 0.0;
            xGrid2 = 0.0;
            yGrid2 = 0.0;
        }

        #region Methods
        /// <summary>[PENDING] Temporally 10x10 terrain definition</summary>
        public void BuildTERRAIN()
        {
            if (grid == null)
            {
                grid = new double[100];
                grid[00] = 0.32; grid[01] = 0.28; grid[02] = 0.27; grid[03] = 0.26;
grid[04] = 0.26; grid[05] = 0.20; grid[06] = 0.03; grid[07] = 0.03; grid[08] = 0.03;
grid[09] = 0.03;
                grid[10] = 0.30; grid[11] = 0.28; grid[12] = 0.26; grid[13] = 0.25;
grid[14] = 0.24; grid[15] = 0.18; grid[16] = 0.03; grid[17] = 0.05; grid[18] = 0.03;
grid[19] = 0.03;
                grid[20] = 0.30; grid[21] = 0.31; grid[22] = 0.25; grid[23] = 0.21;
grid[24] = 0.20; grid[25] = 0.15; grid[26] = 0.09; grid[27] = 0.05; grid[28] = 0.03;
grid[29] = 0.03;
```

```
                grid[30] = 0.29; grid[31] = 0.30; grid[32] = 0.23; grid[33] = 0.30;
grid[34] = 0.16; grid[35] = 0.12; grid[36] = 0.06; grid[37] = 0.05; grid[38] = 0.03;
grid[39] = 0.03;
                grid[40] = 0.28; grid[41] = 0.21; grid[42] = 0.22; grid[43] = 0.15;
grid[44] = 0.15; grid[45] = 0.11; grid[46] = 0.08; grid[47] = 0.05; grid[48] = 0.03;
grid[49] = 0.05;
                grid[50] = 0.26; grid[51] = 0.19; grid[52] = 0.17; grid[53] = 0.16;
grid[54] = 0.13; grid[55] = 0.09; grid[56] = 0.17; grid[57] = 0.15; grid[58] = 0.09;
grid[59] = 0.07;
                grid[60] = 0.25; grid[61] = 0.19; grid[62] = 0.19; grid[63] = 0.14;
grid[64] = 0.18; grid[65] = 0.20; grid[66] = 0.15; grid[67] = 0.16; grid[68] = 0.11;
grid[69] = 0.12;
                grid[70] = 0.21; grid[71] = 0.21; grid[72] = 0.19; grid[73] = 0.19;
grid[74] = 0.22; grid[75] = 0.16; grid[76] = 0.20; grid[77] = 0.18; grid[78] = 0.12;
grid[79] = 0.22;
                grid[80] = 0.22; grid[81] = 0.21; grid[82] = 0.20; grid[83] = 0.23;
grid[84] = 0.18; grid[85] = 0.21; grid[86] = 0.16; grid[87] = 0.15; grid[88] = 0.19;
grid[89] = 0.23;
                grid[90] = 0.23; grid[91] = 0.22; grid[92] = 0.24; grid[93] = 0.19;
grid[94] = 0.18; grid[95] = 0.22; grid[96] = 0.18; grid[97] = 0.20; grid[98] = 0.16;
grid[99] = 0.18;
            }
        }

        /// <summary>[PENDING] Temporally terrain redrawing. Real scale</summary>
        public void RedrawTERRAIN()
        {
            LoadTERRAIN("cata200");
            visor.layer.LoadPicture("Castellar.png"); // Terreno "Castellar_grid.png"

            #region PUSH MATRIX
            Gl.glPushMatrix();
            Gl.glScaled(visor.GlobalScale, visor.GlobalScale, visor.GlobalScale *
visor.ScaleAlt);
            Gl.glTranslated(offsetX, offsetY, offsetZ);
            for (double x = 0.0; x < wi; x += 1.0)
            {
                for (double y = 0.0; y < he; y += 1.0)
                {
                    Gl.glBegin(Gl.GL_QUADS);
                    Gl.glTexCoord2d((x + 0.0) / wi, -(y + 0.0) / he);
Gl.glVertex3d((offsetX + x + offset) * scale, (offsetY + y + offset) * scale, (offsetZ +
grid[(int)(x + y * (wi + 1))]) * scale);
                    Gl.glTexCoord2d((x + 1.0) / wi, -(y + 0.0) / he);
Gl.glVertex3d((offsetX + x + 1 + offset) * scale, (offsetY + y + offset) * scale,
(offsetZ + grid[(int)((x + 1) + y * (wi + 1))]) * scale);
                    Gl.glTexCoord2d((x + 1.0) / wi, -(y + 1.0) / he);
Gl.glVertex3d((offsetX + x + 1 + offset) * scale, (offsetY + y + 1 + offset) * scale,
(offsetZ + grid[(int)((x + 1) + (y + 1) * (wi + 1))]) * scale);
                    Gl.glTexCoord2d((x + 0.0) / wi, -(y + 1.0) / he);
Gl.glVertex3d((offsetX + x + offset) * scale, (offsetY + y + 1 + offset) * scale,
(offsetZ + grid[(int)(x + (y + 1) * (wi + 1))]) * scale);
                    Gl.glEnd();
                }
            }
            Gl.glPopMatrix();
            #endregion PUSH MATRIX

            visor.layer.DefaultPicture();
        }
        /// <summary>[PENDING] Load Digital Elevation Model</summary>
        /// <param name="fileName">The file name without path file and without
extension</param>
        public void LoadTERRAIN(string fileName)
        {
            if (!isDEMLoaded && !map.isDEMCorrupt && map.isDEMExists && !isDEMLoading)
            {
                name = visor.Path + map.FullPath(fileName);
                if (!File.Exists(name))
                {
                    System.Console.WriteLine("fichero " + name + " no encontrado");
                    map.isDEMExists = false;
                }
                else
                {
                    isDEMLoading = true;
                    // starts loading DEM
```

```
                    ThreadStart ts = new ThreadStart(Run);
                    th = new Thread(ts);
                    th.Start();
                }
            }
        }

        private void Run()
        {
            //System.Console.WriteLine("proceso iniciado");
            try
            {
                string l;
                string[] args = null;
                char[] s;
                s = new char[1];
                s[0] = ' ';
                bool isGrid = false;
                int i = 0;
                StreamReader sr = new StreamReader(name);
                if (sr != null)
                {
                    #region Cargar cabecera
                    while (((l = sr.ReadLine()) != null) && (!isGrid))
                    {
                        if (l.EndsWith(" "))
                        {
                            l.TrimEnd(s);
                        }
                        args = l.Split(s);
                        if(!map.ReadHeader(args))
                        {
                            isGrid = true;
                        }
                    }
                    #endregion Cargar cabecera

                    #region Cargar grid
                    // Preproceso de args
                    map.grid = new UTM[map.NCOLS * map.NROWS];
                    do
                    {
                        foreach (string e in args)
                        {
                            if (!e.Equals(""))
                            {
                                // Se presupone UTMZone 'T'
                                map.grid[i] = new UTM(map.Yll(i), map.Xll(i),
Convert.ToDouble(e), "T");
                                i++;
                            }
                        }
                        l = sr.ReadLine();
                        args = l.Split(s);
                    }
                    while (!sr.EndOfStream);
                    #endregion Cargar grid
                    isDEMLoaded = true;
                }
                else
                {
                    System.Console.WriteLine("File " + name + " is corrupt");
                    map.isDEMCorrupt = true;
                }
            }
            catch (Exception e)
            {
                System.Console.WriteLine("; " + e.Message);
                visor.globalExceptionMessage = "Exception '" + e.Message + "' when
loading DEM: " + name;
                map.isDEMCorrupt = true;
            }
            isDEMLoading = false;
            scale = 50;
            //System.Console.WriteLine("proceso terminado");
            th.Abort();
        }
```

```
        #endregion Methods
    }
}
```

VisorControl.Addons.DEMmap

```
public DEMmap()
public double Yll(int i)
public double Xll(int i)
public string FullPath(string fileName)
public bool ReadHeader(string[] args)
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace VisorControl.Addons
{
    /// <summary>[PENDING] Contains the DEM (Digital Elevation Model) inside the UTM
grid. DTM and USGS supported</summary>
    /// <remarks>
    /// <para>Addons/DEMmap. AutoNAV4D 4.1 Revisión 93</para>
    /// <para>©Susana Romero</para>
    /// <para>Esta clase está protegida por la licencia BSD de código abierto.</para>
    /// <para>0.Puedes ejecutar el programa con cualquier propósito.</para>
    /// <para>1.Puedes estudiar y/o modificar el programa.</para>
    /// <para>2.Puedes copiar el programa de manera que puedas ayudar a tu
vecino.</para>
    /// <para>3.Puedes mejorar el programa, y hacer públicas las mejoras, de forma que
se beneficie toda la comunidad.</para>
    /// <para>...</para>
    /// <para>Contenedor de las coordenada UTM del DEM (Digital Elevation Model).
Soporta ficheros DTM y USGS.</para>
    /// <para>...</para>
    /// </remarks>
    [Serializable]
    public class DEMmap
    {
        #region Fields
        /// <summary>Available formats for DEM</summary>
        public enum Header { DTM, USGS };
        public Header headerType;
        private DTMHeader headerDTM;
        private USGSHeader headerUSGS;
        public int NCOLS;
        public int NROWS;
        public UTM[] grid;
        public bool isDEMCorrupt;
        public bool isDEMExists;
        #endregion Fields

        public DEMmap()
        {
            headerType = Header.DTM;
            headerDTM = new DTMHeader();
            headerUSGS = new USGSHeader();
            NCOLS = 0;
            NROWS = 0;
            grid = null;
            isDEMCorrupt = false;
            isDEMExists = true;
        }

        #region Methods

        /// <summary>[VALIDATED] Returns northing coordinate depending on DEM size for a
given reading index</summary>
        /// <param name="i">Reading index</param>
        /// <returns>Northing coordinate. Retuns 0.0 if heading is empty</returns>
        public double Yll(int i)
        {
            if (headerType == Header.USGS) // The slower first
            {
                return headerUSGS.Yll(i);
            }
            else if (headerType == Header.DTM) // The faster second
            {
                return headerDTM.Yll(i);
```

```
            }
            else
            {
                return 0.0;
            }
        }

        /// <summary>[VALIDATED] Returns easting coordinate depending on DTM size for a
given reading index</summary>
        /// <param name="i">Reading index</param>
        /// <returns>Easting coordinate. Retuns 0.0 if heading is empty</returns>
        public double Xll(int i)
        {
            if (headerType == Header.USGS) // The slower first
            {
                return headerUSGS.Xll(i);
            }
            else if (headerType == Header.DTM) // The faster second
            {
                return headerDTM.Xll(i);
            }
            else
            {
                return 0.0;
            }
        }

        /// <summary>[VALIDATED] Returns the normalized file name with the DEM folder
and the correct extension</summary>
        /// <param name="fileName">The file name with out extension. Changes the type if
extension is included</param>
        /// <returns>The normalized file name. Returns empty if no type is
selected</returns>
        public string FullPath(string fileName)
        {
            // Detects the extension if any
            if (fileName.ToLower().EndsWith(".usgs")) // The slower first
            {
                headerType = Header.USGS;
                return @"Groups/DEM/" + fileName;
            }
            else if (fileName.ToLower().EndsWith(".dtm")) // The faster second
            {
                headerType = Header.DTM;
                return @"Groups/DEM/" + fileName;
            }
            // Normalize the name file depending on the header Type
            if (headerType == Header.USGS) // The slower first
            {
                return @"Groups/DEM/" + fileName + ".usgs";
            }
            else if (headerType == Header.DTM) // The faster second
            {
                return @"Groups/DEM/" + fileName + ".dtm";
            }
            else
            {
                return "";
            }
        }

        /// <summary>[PENDING] Parses a heading sentence and detects the grid
start</summary>
        /// <param name="args">sentences to be parsed</param>
        /// <returns>True if parse is valid</returns>
        public bool ReadHeader(string[] args)
        {
            if (headerType == Header.USGS) // The slower first
            {
                bool r = headerUSGS.ReadHeader(args);
                NCOLS = headerUSGS.NCOLS;
                NROWS = headerUSGS.NROWS;
                return r;
            }
            else if (headerType == Header.DTM) // The faster second
            {
                bool r = headerDTM.ReadHeader(args);
```

```
                NCOLS = headerDTM.NCOLS;
                NROWS = headerDTM.NROWS;
                return r;
            }
            else
            {
                return false;
            }
        }
        #endregion Methods
    }
}
```

VisorControl.Addons. DTMHeader

```
    public DTMHeader()
    public double Yll(int i)
    public double Xll(int i)
    public bool ReadHeader(string[] args)
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace VisorControl.Addons
{
    /// <summary>[PENDING] File terrain header for DTM format</summary>
    /// <remarks>
    /// <para>Addons/DTMHeader. AutoNAV4D 4.1 Revisión 93</para>
    /// <para>©Susana Romero</para>
    /// <para>Esta clase está protegida por la licencia BSD de código abierto.</para>
    /// <para>0.Puedes ejecutar el programa con cualquier propósito.</para>
    /// <para>1.Puedes estudiar y/o modificar el programa.</para>
    /// <para>2.Puedes copiar el programa de manera que puedas ayudar a tu
vecino.</para>
    /// <para>3.Puedes mejorar el programa, y hacer públicas las mejoras, de forma que
se beneficie toda la comunidad.</para>
    /// <para>...</para>
    /// <para>Contenedor de los datos de la cabecera del DTM (Digital Terrain
Model)</para>
    /// <para>...</para>
    /// <para>DEFINICIÓN DEL FORMATO ASCII de Arc/Info (.DTM)</para>
    /// <para>Los mapas de catalunya los hemos obtenido de
http://www.icc.es/web/content/ca/prof/cartografia/descarrega.html</para>
    /// <para>Formato ASCII de Arc/Info</para>
    /// <para>El formato de los ficheros .dtm es el formato 'grid ASCII' de Arc/Info
pero que se puede leer con la mayoría de programas que puedan leer modelos de malla
regular en formato ASCII.</para>
    /// <para>Hay una cabecera con el número de columnas, filas, coordenadas del centro
del pixel sur-oeste, paso de malla y símbolo de cota inexistente.</para>
    /// <para>NCOLS xxx</para>
    /// <para>NROWS xxx</para>
    /// <para>XLLCENTER xxx</para>
    /// <para>YLLCENTER xxx</para>
    /// <para>CELLSIZE xxx</para>
    /// <para>NODATA_VALUE xxx</para>
    /// <para>A continuación vienen las cotas por filas de Norte a Sur y cada fila de
Oeste a Este. Las cotas vienen dadas en metros y separadas por blancos. Dependiendo de
como se haya generado puede haber saltos de línea o no y se recomienda leerlas con
lectura de flujo 'stream' con formato libre.</para>
    /// <para>El sistema de referencia es el oficial ED50 proyección UTM huso 31 y las
cotas son ortométricas. La altura ortométrica es la distancia en vertical entre la
superficie física del planeta y la superficie del geoide. Las alturas ortométricas (H)
se relacionan con las alturas geodésicas (N) como H=h-N sonde h es la la altura
elipsoidal.</para>
    /// </remarks>
    [Serializable]
    public class DTMHeader : IHeader
    {
        public int NCOLS;
        public int NROWS;
        public double XLLCENTER;
        public double YLLCENTER;
        public double CELLSIZE;
        public double NODATA_VALUE;

        public DTMHeader()
        {
            NCOLS = 0;
            NROWS = 0;
            XLLCENTER = 0;
            YLLCENTER = 0;
            CELLSIZE = 0;
            NODATA_VALUE = 0;
        }
```

```
        #region Methods

        /// <summary>[pending] Returns northing coordinate depending on DTM size from
read index</summary>
        /// <param name="i">Read index</param>
        /// <returns>Northing coordinate. Retuns 0 if heading not available</returns>
        public double Yll(int i)
        {
            if (NROWS == 0)
            {
                return 0;
            }
            else
            {
                return ((double)NROWS * CELLSIZE - (double)(i / NROWS) * CELLSIZE);
            }
        }

        /// <summary>[pending] Returns easting coordinate depending on DTM size from
read index</summary>
        /// <param name="i">Read index</param>
        /// <returns>Easting coordinate. Retuns 0 if heading not available</returns>
        public double Xll(int i)
        {
            if (NCOLS == 0)
            {
                return 0;
            }
            else
            {
                return ((double)NCOLS * CELLSIZE - ((((double)i / NCOLS) - (double)(i /
NCOLS)) * NCOLS * CELLSIZE));
            }
        }

        public bool ReadHeader(string[] args)
        {
            try
            {
                if (args[0].Equals("NCOLS"))
                {
                    NCOLS = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("NROWS"))
                {
                    NROWS = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("XLLCENTER"))
                {
                    XLLCENTER = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("YLLCENTER"))
                {
                    YLLCENTER = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("CELLSIZE"))
                {
                    CELLSIZE = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("NODATA_VALUE"))
                {
                    NODATA_VALUE = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else
                {
                    return false;
                }
            }
            catch (Exception e)
            {
```

```
                System.Console.WriteLine("Exception '" + e.Message + "' when loading
DEM: ");
                return false;
            }
        }
        #endregion Methods
    }
}
```

VisorControl.Addons. USGSHeader

```
public USGSHeader()
public double Xll(int i)
public double Yll(int i)
public bool ReadHeader(string[] args)
```

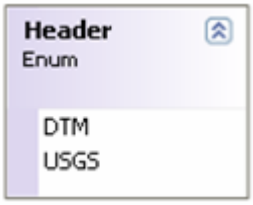
```
using System;
using System.Collections.Generic;
using System.Text;

namespace VisorControl.Addons
{
    /// <summary>[PENDING] File terrain header for USGS format</summary>
    /// <remarks>
    /// <para>Addons/USGSHeader. AutoNAV4D 4.1 Revisión 93</para>
    /// <para>©Susana Romero</para>
    /// <para>Esta clase está protegida por la licencia BSD de código abierto.</para>
    /// <para>0.Puedes ejecutar el programa con cualquier propósito.</para>
    /// <para>1.Puedes estudiar y/o modificar el programa.</para>
    /// <para>2.Puedes copiar el programa de manera que puedas ayudar a tu
vecino.</para>
    /// <para>3.Puedes mejorar el programa, y hacer públicas las mejoras, de forma que
se beneficie toda la comunidad.</para>
    /// <para>...</para>
    /// <para>Contenedor de los datos de la cabecera del DTM (Digital Terrain
Model)</para>
    /// <para>...</para>
    /// <para>DEFINICIÓN DEL FORMATO ASCII de Arc/Info (.USGS)</para>
    /// <para>Los mapas globales los hemos obtenido de
http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html</para>
    /// <para>Formato ASCII está especifcado en
http://edc.usgs.gov/products/elevation/gtopo30/README.html</para>
    /// <para>La cabecera contiene:</para>
    /// <para>BYTEORDER orden del byte en el cual se guardan los valores de los píxels
de la imagen. M = Motorola byte order (primero el byte más significativo) </para>
    /// <para>LAYOUT  organización de las bandas en el archivo. BIL = band interleaved
by line (nota: el DEM es una imagen de banda única)</para>
    /// <para>NROWS  número de filas de la imagen</para>
    /// <para>NCOLS  número de columnas de las imagen</para>
    /// <para>NBANDS número de las bandas espectrales de la imagen (1 para DEM)</para>
    /// <para>NBITS número de bits por píxel (16 para DEM) </para>
    /// <para>BANDROWBYTES número de bytes por banda por fila (el doble del número de
columnas del DEM de 16 bits) </para>
    /// <para>TOTALROWBYTES número total de bytes de datos por fila (el doble del número
de columnas para un DEM de 16 bits de banda única) </para>
    /// <para>BANDGAPBYTES el número de bytes entre las bandas en una imagen con formato
BSQ(0 para DEM) </para>
    /// <para>NODATA símbolo de cota inexistente </para>
    /// <para>ULXMAP longitud del centro del píxel superior izquierdo (grados decimales)
</para>
    /// <para>ULYMAP latitud del centro del píxel superior izquierdo (grados decimales)
</para>
    /// <para>XDIM dimensión del píxel x en unidades geográficas (grados decimales)
</para>
    /// <para>YDIM dimensión del píxel y en unidades geográficas (grados decimales)
</para>
    /// <para>...</para>
    /// <para>El sistema de referencia es el WGS84.</para>
    /// </remarks>
    [Serializable]
    public class USGSHeader : IHeader
    {
        public bool isBYTEORDER;
        public bool isLAYOUT;
        public int NROWS;
        public int NCOLS;
        public int NBANDS;
        public int NBITS;
        public int BANDROWBYTES;
        public int TOTALROWBYTES;
        public int BANDGAPBYTES;
        public int NODATA;
        public double ULXMAP;
```

```
        public double ULYMAP;
        public double XDIM;
        public double YDIM;

        public USGSHeader()
        {
            isBYTEORDER = false;
            isLAYOUT = true;
            NROWS = 0;
            NCOLS = 0;
            NBANDS = 0;
            NBITS = 0;
            BANDROWBYTES = 0;
            TOTALROWBYTES = 0;
            BANDGAPBYTES = 0;
            NODATA = 0;
            ULXMAP = 0;
            ULYMAP = 0;
            XDIM = 0;
            YDIM = 0;
        }

        public double Xll(int i)
        {
            return 0.0;
        }

        public double Yll(int i)
        {
            return 0.0;
        }

        public bool ReadHeader(string[] args)
        {
            try
            {
                if (args[0].Equals("NROWS"))
                {
                    NROWS = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("NCOLS"))
                {
                    NCOLS = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("NBANDS"))
                {
                    NBANDS = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("NBITS"))
                {
                    NBITS = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("BANDROWBYTES"))
                {
                    BANDROWBYTES = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("TOTALROWBYTES"))
                {
                    TOTALROWBYTES = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("BANDGAPBYTES"))
                {
                    BANDGAPBYTES = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("NODATA"))
                {
                    NODATA = Convert.ToInt32(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("ULXMAP"))
```

```
                {
                    ULXMAP = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("ULYMAP"))
                {
                    ULYMAP = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("ULYMAP"))
                {
                    ULYMAP = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else if (args[0].Equals("YDIM"))
                {
                    YDIM = Convert.ToDouble(args[args.Length - 1]);
                    return true;
                }
                else
                {
                    return false;
                }
            }
            catch (Exception e)
            {
                System.Console.WriteLine("Exception '" + e.Message + "' when loading
DEM: ");
                return false;
            }
        }
    }
}
```

VisorControlInterface. IHeader

```
    double Xll(int i)
    double Yll(int i)
    bool ReadHeader(string[] args)
```




```
using System;
using System.Collections.Generic;
using System.Text;

namespace VisorControl
{
    /// <summary>[VALIDATED] DEM header interface</summary>
    /// <remarks>
    /// <para>IHeader. AutoNAV4D 4.1 Revisión 93</para>
    /// <para>©Susana Romero.</para>
    /// <para>Esta clase está protegida por la licencia BSD de código abierto.</para>
    /// <para>0.Puedes ejecutar el programa con cualquier propósito.</para>
    /// <para>1.Puedes estudiar y/o modificar el programa.</para>
    /// <para>2.Puedes copiar el programa de manera que puedas ayudar a tu
vecino.</para>
    /// <para>3.Puedes mejorar el programa, y hacer públicas las mejoras, de forma que
se beneficie toda la comunidad.</para>
    /// <para>...</para>
    /// <para>Interfaz de cabecera de mapas DEM.</para>
    /// <para>En la implementación del interfaz deben contemplarse valores por defecto
en caso de no ser proporcionados o descartar valores en el caso de no estar
asignados.</para>
    /// </remarks>
    public interface IHeader
    {
        double Xll(int i);
        double Yll(int i);
        bool ReadHeader(string[] args);
    }
}
```

VisorControl.Addons. UTM

```
using System;
using System.Collections.Generic;
using System.Text;

namespace VisorControl.Addons
{
    /// <summary>[PENDING] Universal Transverse Mercator coordinates (northing, easting,
altitude) in a NEU reference system. Position over the map in meters</summary>
    /// <remarks>
    /// <para>Addons/UTM. AutoNAV4D 4.1 Revisión 93</para>
    /// <para>©Joshua Tristancho M.</para>
    /// <para>Esta clase está protegida por la licencia BSD de código abierto.</para>
    /// <para>0.Puedes ejecutar el programa con cualquier propósito.</para>
    /// <para>1.Puedes estudiar y/o modificar el programa.</para>
    /// <para>2.Puedes copiar el programa de manera que puedas ayudar a tu
vecino.</para>
    /// <para>3.Puedes mejorar el programa, y hacer públicas las mejoras, de forma que
se beneficie toda la comunidad.</para>
    /// <para>...</para>
    /// </remarks>
    [Serializable]
    public class UTM
    {
        /// <summary>Northing 80S a 84N in meters  [0 to 9328000]</summary>
        public double n;
        /// <summary>Easting 180W a 180E in meters [0 to 10000000]</summary>
        public double e;
        /// <summary>Altitude or high over mean sea level in meters [-rPLANET to
3.4E+38]</summary>
        public double u;
        /// <summary>Zone number '00N' (01 to 60)</summary>
        public int zoneNumber;
        /// <summary>Desigantor code '00N' (C to X except I and O)</summary>
        public string designator;
        /// <summary>UTM zone code '00N' (1 to 60 and C to X except I and O)</summary>
        string zone;
        public UTM()
        {
            n = 0;
            e = 500000;
            u = 0;
            zoneNumber = 31;
            designator = "N";
            zone = zoneNumber.ToString().PadLeft(2, '0').ToUpper() + designator;
        }
        public UTM(double n, double e, double u, string UTMZone)
        {
            this.n = n;
            this.e = e;
            this.u = u;
            if (UTMZone.Length > 2)
            {
                this.zoneNumber = Convert.ToInt32(UTMZone.Substring(2, 1));
                this.designator = UTMZone.Substring(0, 2).ToUpper();
            }
            else if (UTMZone.Length > 1)
            {
                this.zoneNumber = Convert.ToInt32(UTMZone.Substring(1, 1));
                this.designator = UTMZone.Substring(0, 1).ToUpper();
            }
            else
            {
                this.zoneNumber = 0;
                this.designator = "Z";
                UTMZone = "00Z"; // Zona no válida
            }
            this.zone = zoneNumber.ToString().PadLeft(2, '0').ToUpper() + designator;
        }
        /// <summary>[PENDING] Universal Transverse Mercator nomenclator</summary>
        /// <returns>The normalized UTM nomenclature</returns>
        /// <remarks>
        /// <para>Devuelve el nombre normalizado a partir de las coordenadas UTM.</para>
```

```
        /// <para>UTM es una proyección cilíndrica conforme. Las líneas loxodrómicas se
representan como líneas rectas sobre el plano. solo se representa la region entre los
paralelos 84ºN y 80ºS.</para>
        /// <para>Se divide la Tierra en 60 husos de 6º grados de longitud y se enumeran
del 1 al 60 el intervalo [-180º, 180) empeando por 180W.</para>
        /// <para>Se divide la Tierra en 20 zonas de 8º grados de Latitud, que se
denominan con letras desde la C hasta la X excluyendo las letras 'I' y 'O'. De la 'N' en
adelante están en el emisferio Norte.</para>
        /// <para>Las zonas polares no están consideradas en este sistema (Ver sistema
de coordenadas UPS) Las dos excepciones son la zona 31V y la 32V.</para>
        /// <para>http://erg.usgs.gov/isb/pubs/factsheets/fs07701.html</para>
        /// </remarks>
        public string ToNomenclature()
        {
            return "30S3546784891567";
        }
    }
}
```